

讲堂 > Linux性能优化实战 > 文章详情

07 | 案例篇：系统中出现大量不可中断进程和僵尸进程怎么办？（上）

2018-12-05 倪朋飞



07 | 案例篇：系统中出现大量不可中断进程和僵尸进程怎么办？（上）

朗读人：冯永吉 12'35" | 11.54M

你好，我是倪朋飞。

上一节，我用一个 Nginx+PHP 的案例，给你讲了服务器 CPU 使用率高的分析和应对方法。这里你一定要记得，当碰到无法解释的 CPU 使用率问题时，先要检查一下是不是短时应用在捣鬼。

短时应用的运行时间比较短，很难在 top 或者 ps 这类展示系统概要和进程快照的工具中发现，你需要使用记录事件的工具来配合诊断，比如 execsnoop 或者 perf top。

这些思路你不用刻意去背，多练习几次，多在操作中思考，你便能灵活运用。

另外，我们还讲到 CPU 使用率的类型。除了上一节提到的用户 CPU 之外，它还包括系统 CPU（比如上下文切换）、等待 I/O 的 CPU（比如等待磁盘的响应）以及中断 CPU（包括软中断和硬中断）等。

我们已经在上下文切换的文章中，一起分析了系统 CPU 使用率高的问题，剩下的等待 I/O 的 CPU 使用率（以下简称为 iowait）升高，也是最常见的一个服务器性能问题。今天我们就来看一个多进程 I/O 的案例，并分析这种情况。

进程状态

当 iowait 升高时，进程很可能因为得不到硬件的响应，而长时间处于不可中断状态。从 ps 或者 top 命令的输出中，你可以发现它们都处于 D 状态，也就是不可中断状态（Uninterruptible Sleep）。既然说到了进程的状态，进程有哪些状态你还记得吗？我们先来回顾一下。

top 和 ps 是最常用的查看进程状态的工具，我们就从 top 的输出开始。下面是一个 top 命令输出的示例，S 列（也就是 Status 列）表示进程的状态。从这个示例里，你可以看到 R、D、Z、S、I 等几个状态，它们分别是什么意思呢？

1 \$ top

	PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3	28961	root	20	0	43816	3148	4040	R	3.2	0.0	0:00.01	top
4	620	root	20	0	37280	33676	908	D	0.3	0.4	0:00.01	app
5	1	root	20	0	160072	9416	6752	S	0.0	0.1	0:37.64	systemd
6	1896	root	20	0	0	0	0	Z	0.0	0.0	0:00.00	devapp
7	2	root	20	0	0	0	0	S	0.0	0.0	0:00.10	kthreadd
8	4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/0:0H
9	6	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	mm_percpu_wq
10	7	root	20	0	0	0	0	S	0.0	0.0	0:06.37	ksoftirqd/0

复制代码

我们挨个来看一下：

- **R** 是 Running 或 Runnable 的缩写，表示进程在 CPU 的就绪队列中，正在运行或者正在等待运行。
- **D** 是 Disk Sleep 的缩写，也就是不可中断状态睡眠（Uninterruptible Sleep），一般表示进程正在跟硬件交互，并且交互过程不允许被其他进程或中断打断。
- **Z** 是 Zombie 的缩写，如果你玩过“植物大战僵尸”这款游戏，应该知道它的意思。它表示僵尸进程，也就是进程实际上已经结束了，但是父进程还没有回收它的资源（比如进程的描述符、PID 等）。
- **S** 是 Interruptible Sleep 的缩写，也就是可中断状态睡眠，表示进程因为等待某个事件而被系统挂起。当进程等待的事件发生时，它会被唤醒并进入 R 状态。
- **I** 是 Idle 的缩写，也就是空闲状态，用在不可中断睡眠的内核线程上。前面说了，硬件交互导致的不可中断进程用 D 表示，但对某些内核线程来说，它们有可能实际上并没有任何负

载，用 Idle 正是为了区分这种情况。要注意，D 状态的进程会导致平均负载升高，I 状态的进程却不会。

当然了，上面的示例并没有包括进程的所有状态。除了以上 5 个状态，进程还包括下面这 2 个状态。

第一个是 **T 或者 t**，也就是 Stopped 或 Traced 的缩写，表示进程处于暂停或者跟踪状态。

向一个进程发送 SIGSTOP 信号，它就会因响应这个信号变成暂停状态（Stopped）；再向它发送 SIGCONT 信号，进程又会恢复运行（如果进程是终端里直接启动的，则需要你用 fg 命令，恢复到前台运行）。

而当你用调试器（如 gdb）调试一个进程时，在使用断点中断进程后，进程就会变成跟踪状态，这其实也是一种特殊的暂停状态，只不过你可以用调试器来跟踪并按需要控制进程的运行。

另一个是 **X**，也就是 Dead 的缩写，表示进程已经消亡，所以你不会在 top 或者 ps 命令中看到它。

了解了这些，我们再回到今天的主题。先看不可中断状态，这其实是为了保证进程数据与硬件状态一致，并且正常情况下，不可中断状态在很短时间内就会结束。所以，短时的不可中断状态进程，我们一般可以忽略。

但如果系统或硬件发生了故障，进程可能会在不可中断状态保持很久，甚至导致系统中出现大量不可中断进程。这时，你就得注意下，系统是不是出现了 I/O 等性能问题。

再看僵尸进程，这是多进程应用很容易碰到的问题。正常情况下，当一个进程创建了子进程后，它应该通过系统调用 wait() 或者 waitpid() 等待子进程结束，回收子进程的资源；而子进程在结束时，会向它的父进程发送 SIGCHLD 信号，所以，父进程还可以注册 SIGCHLD 信号的处理函数，异步回收资源。

如果父进程没这么做，或是子进程执行太快，父进程还没来得及处理子进程状态，子进程就已经提前退出，那这时的子进程就会变成僵尸进程。换句话说，父亲应该一直对儿子负责，善始善终，如果不作为或者跟不上，都会导致“问题少年”的出现。

通常，僵尸进程持续的时间都比较短，在父进程回收它的资源后就会消亡；或者在父进程退出后，由 init 进程回收后也会消亡。

一旦父进程没有处理子进程的终止，还一直保持运行状态，那么子进程就会一直处于僵尸状态。大量的僵尸进程会用尽 PID 进程号，导致新进程不能创建，所以这种情况一定要避免。

案例分析

接下来，我将用一个多进程应用的案例，带你分析大量不可中断状态和僵尸状态进程的问题。这个应用基于 C 开发，由于它的编译和运行步骤比较麻烦，我把它打包成了一个 [Docker 镜像](#)。这样，你只需要运行一个 Docker 容器就可以得到模拟环境。

你的准备

下面的案例仍然基于 Ubuntu 18.04，同样适用于其他的 Linux 系统。我使用的案例环境如下所示：

- 机器配置：2 CPU，8GB 内存
- 预先安装 docker、sysstat、dstat 等工具，如 `apt install docker.io dstat sysstat`

这里，dstat 是一个新的性能工具，它吸收了 vmstat、iostat、ifstat 等几种工具的优点，可以同时观察系统的 CPU、磁盘 I/O、网络以及内存使用情况。

接下来，我们打开一个终端，SSH 登录到机器上，并安装上述工具。

注意，以下所有命令都默认以 root 用户运行，如果你用普通用户身份登陆系统，请运行 `sudo su root` 命令切换到 root 用户。

如果安装过程有问题，你可以先上网搜索解决，实在解决不了的，记得在留言区向我提问。

温馨提示：案例应用的核心代码逻辑比较简单，你可能一眼就能看出问题，但实际生产环境中的源码就复杂多了。所以，我依旧建议，操作之前别看源码，避免先入为主，而要把它当成一个黑盒来分析，这样你可以更好地根据现象分析问题。你姑且当成你工作中的一次演练，这样效果更佳。

操作和分析

安装完成后，我们首先执行下面的命令运行案例应用：

```
1 $ docker run --privileged --name=app -itd feisky/app:iowait
```

[复制代码](#)

然后，输入 `ps` 命令，确认案例应用已正常启动。如果一切正常，你应该可以看到如下所示的输出：

```
1 $ ps aux | grep /app
2 root      4009  0.0  0.0   4376  1008 pts/0    Ss+  05:51   0:00 /app
3 root      4287  0.6  0.4  37280 33660 pts/0    D+   05:54   0:00 /app
4 root      4288  0.6  0.4  37280 33668 pts/0    D+   05:54   0:00 /app
```

[复制代码](#)

从这个界面，我们可以发现多个 app 进程已经启动，并且它们的状态分别是 Ss+ 和 D+。其中，S 表示可中断睡眠状态，D 表示不可中断睡眠状态，我们在前面刚学过，那后面的 s 和 + 是什么意思呢？不知道也没关系，查一下 `man ps` 就可以。现在记住，s 表示这个进程是一个会话的领导进程，而 + 表示前台进程组。

这里又出现了两个新概念，**进程组**和**会话**。它们用来管理一组相互关联的进程，意思其实很好理解。

- 进程组表示一组相互关联的进程，比如每个子进程都是父进程所在组的成员；
- 而会话是指共享同一个控制终端的一个或多个进程组。

比如，我们通过 SSH 登录服务器，就会打开一个控制终端（TTY），这个控制终端就对应一个会话。而我们在终端中运行的命令以及它们的子进程，就构成了一个个的进程组，其中，在后台运行的命令，构成后台进程组；在前台运行的命令，构成前台进程组。

明白了这些，我们再用 `top` 看一下系统的资源使用情况：

[复制代码](#)

```
1 # 按下数字 1 切换到所有 CPU 的使用情况，观察一会儿按 Ctrl+C 结束
2 $ top
3 top - 05:56:23 up 17 days, 16:45, 2 users, load average: 2.00, 1.68, 1.39
4 Tasks: 247 total, 1 running, 79 sleeping, 0 stopped, 115 zombie
5 %Cpu0 : 0.0 us, 0.7 sy, 0.0 ni, 38.9 id, 60.5 wa, 0.0 hi, 0.0 si, 0.0 st
6 %Cpu1 : 0.0 us, 0.7 sy, 0.0 ni, 4.7 id, 94.6 wa, 0.0 hi, 0.0 si, 0.0 st
7 ...
8
9      PID USER      PR  NI   VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
10    4340 root        20   0   44676   4048   3432 R   0.3   0.0   0:00.05 top
11    4345 root        20   0   37280  33624    860 D   0.3   0.0   0:00.01 app
12    4344 root        20   0   37280  33624    860 D   0.3   0.4   0:00.01 app
13        1 root        20   0 160072   9416   6752 S   0.0   0.1   0:38.59 systemd
14 ...
```

从这里你能看出什么问题吗？细心一点，逐行观察，别放过任何一个地方。忘了哪行参数意思的话，也要及时返回去复习。

好的，如果你已经有了答案，那就继续往下走，看看跟我找的问题是否一样。这里，我发现了四个可疑的地方。

- 先看第一行的平均负载（Load Average），过去 1 分钟、5 分钟和 15 分钟内的平均负载在依次减小，说明平均负载正在升高；而 1 分钟内的平均负载已经达到系统的 CPU 个数，说明系统很可能已经有了性能瓶颈。

- 再看第二行的 Tasks，有 1 个正在运行的进程，但僵尸进程比较多，而且还在不停增加，说明有子进程在退出时没被清理。
- 接下来看两个 CPU 的使用率情况，用户 CPU 和系统 CPU 都不高，但 iowait 分别是 60.5% 和 94.6%，好像有点儿不正常。
- 最后再看每个进程的情况，CPU 使用率最高的进程只有 0.3%，看起来并不高；但有两个进程处于 D 状态，它们可能在等待 I/O，但光凭这里并不能确定是它们导致了 iowait 升高。

我们把这四个问题再汇总一下，就可以得到很明确的两点：

- 第一点，iowait 太高了，导致系统的平均负载升高，甚至达到了系统 CPU 的个数。
- 第二点，僵尸进程在不断增多，说明有程序没能正确清理子进程的资源。

那么，碰到这两个问题该怎么办呢？结合我们前面分析问题的思路，你先自己想想，动手试试，下节课我来继续“分解”。

小结

今天我们主要通过简单的操作，熟悉了几个必备的进程状态。用我们最熟悉的 ps 或者 top，可以查看进程的状态，这些状态包括运行（R）、空闲（I）、不可中断睡眠（D）、可中断睡眠（S）、僵尸（Z）以及暂停（T）等。

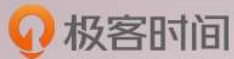
其中，不可中断状态和僵尸状态，是我们今天学习的重点。

- 不可中断状态，表示进程正在跟硬件交互，为了保护进程数据和硬件的一致性，系统不允许其他进程或中断打断这个进程。进程长时间处于不可中断状态，通常表示系统有 I/O 性能问题。
- 僵尸进程表示进程已经退出，但它的父进程还没有回收子进程占用的资源。短暂的僵尸状态我们通常不必理会，但进程长时间处于僵尸状态，就应该注意了，可能有应用程序没有正常处理子进程的退出。

思考

最后，我想请你思考一下今天的课后题，案例中发现的这两个问题，你会怎么分析呢？又应该怎么解决呢？你可以结合前面我们做过的案例分析，总结自己的思路，提出自己的问题。

欢迎在留言区和我讨论，也欢迎把这篇文章分享给你的同事、朋友。我们一起在实战中演练，在交流中进步。



Linux 性能优化实战

10 分钟帮你找到系统瓶颈

倪朋飞

微软资深工程师
Kubernetes 项目维护者



©版权归极客邦科技所有，未经许可不得转载

上一篇 06 | 案例篇：系统的 CPU 使用率很高，但为啥却找不到高 CPU 的应用？

下一篇 08 | 案例篇：系统中出现大量不可中断进程和僵尸进程怎么办？（下）

写留言

精选留言



Johnson

5

遇到有大量的D状态的进程，导致负载到7000多，但是cpu和iowait都不高，除了重启设备还有什么办法解决？

2018-12-05



每天晒白牙

5

【D7打卡】

今天主要学习进程的状态，可以通过ps或top查看进程的状态

R:运行 Running或Runnable的缩写 表示进程在CPU的就绪队列中，正在运行或正在等待运行

I:空闲 Idle的缩写，用在不可中断睡眠的内核线程上。空闲线程不会导致平均负载升高，D状态的会导致平均负载升高

D:不可中断睡眠 Dist Sleep的缩写 表示进程正在跟硬件交互，并且交互过程中不允许被其他进程或中断打断

S:可中断睡眠 Interruptible Sleep的缩写 表示进程因为等待某个事件而被系统挂起。当进程等待的事件发生时，它会被唤醒并进入R状态

Z:僵尸 Zombie缩写 进程已经结束，但父进程还没有回收它的资源（如进程的描述符、PID

等)

T:暂停 Stopped或Traced的缩写，表示进程处于暂停或者跟踪状态

今天实战的结果和老师的出入很大，我的系统是centos的

ps aux | grep /app结果最开始是

```
[root@localhost ~]# ps aux | grep /app
```

```
root 3468 0.0 0.0 4364 380 pts/0 Ss+ 07:38 0:00 /app
```

```
root 3568 12.0 3.3 37268 33036 pts/0 D+ 07:39 0:00 /app
```

```
root 3569 7.0 3.3 37268 33036 pts/0 D+ 07:39 0:00 /app
```

```
root 3571 0.0 0.0 112728 988 pts/4 S+ 07:39 0:00 grep --color=auto /app
```

```
[root@localhost ~]# ps aux | grep /app
```

```
root 3468 0.0 0.0 4364 424 pts/0 Ss+ 07:38 0:00 /app
```

```
root 3590 15.0 3.3 37268 33016 pts/0 R+ 07:39 0:00 /app
```

```
root 3591 15.0 3.3 37268 33016 pts/0 R+ 07:39 0:00 /app
```

```
root 3593 0.0 0.0 112724 988 pts/4 R+ 07:39 0:00 grep --color=auto /app
```

```
[root@localhost ~]# ps aux | grep /app
```

```
root 3468 0.0 0.0 4364 424 pts/0 Ss+ 07:38 0:00 /app
```

```
root 3597 0.0 0.0 112728 988 pts/4 S+ 07:39 0:00 grep --color=auto /app
```

同样用top命令观察，平均负载也不高，然后wa也很低，也没看到处于D状态的进程。符合的只有处于僵尸状态的进程比较多

```
[root@localhost ~]# top
```

```
top - 07:41:45 up 2:35, 8 users, load average: 0.00, 0.03, 0.31
```

```
Tasks: 224 total, 1 running, 135 sleeping, 0 stopped, 88 zombie
```

```
%Cpu0 : 0.7 us, 5.8 sy, 0.0 ni, 88.1 id, 4.1 wa, 0.0 hi, 1.4 si, 0.0 st
```

```
%Cpu1 : 1.0 us, 7.1 sy, 0.0 ni, 86.7 id, 2.7 wa, 0.0 hi, 2.4 si, 0.0 st
```

```
KiB Mem : 999720 total, 165196 free, 389676 used, 444848 buff/cache
```

```
KiB Swap: 2097148 total, 1788172 free, 308976 used. 335844 avail Mem
```

和老师反映了情况，老师说可能是案例里I/O线程还不够多，效果不明显，等着老师修改案例重新发镜像，再实验。

2018-12-05



书林

3

每个人的机器配置不一样，所以会出现有的机器iowait不明显，有的机器被打爆。解决办法是用docker cgroup选项对 block io做限制。假设硬盘设备为 /dev/nvme0n1，测试如下：

1. 限制块设备的读写 iops 为 3: `docker run --privileged --name=app9 --device /dev/nvme0n1:/dev/nvme0n1 --device-write-iops /dev/nvme0n1:3 --device-read-iops /dev/nvme0n1:3 -itd feisky/app:iowait-new2`

2. 可以查看host机器 cgroup 已为对应 docker container 添加了相关限制：

...

```
cat /sys/fs/cgroup/blkio/docker/"docker-container-id"/blkio.throttle.write_iops_device
```

```
259:0 3
```

```
cat /sys/fs/cgroup/blkio/docker/"docker-container-id"/blkio.throttle.read_iops_device
```



```
259:0 3
```

```
...
```

```
3.
```

```
...
```

```
docker exec -it "docker-container-id" /bin/bash
```

```
root@4cc5e6c74cc0:/# dd iflag=direct if=/dev/nvme0n1 of=/dev/null bs=1k count=1000
```

```
1000+0 records in
```

```
1000+0 records out
```

```
1024000 bytes (1.0 MB, 1000 KiB) copied, 340.004 s, 3.0 kB/s
```

```
...
```

`dd` 每次从 /dev/nvme0n1 设备读取数据写到 /dev/null 设备，每次读取 1kB，一共1000次，必须为 direct 选项。可以观测到拷贝速度为 3 kB/s，即 1kB * 3，说明cgroup 限制 `blkio.throttle.read_iops_device` 生效。

4. 观察host机器 iowait 已经上去。

```
...
```

```
top - 12:10:22 up 1:25, 1 user, load average: 0.88, 0.81, 0.83
```

```
任务: 780 total, 1 running, 227 sleeping, 0 stopped, 552 zombie
```

```
%Cpu0 : 0.0 us, 0.0 sy, 0.0 ni,100.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 s
```

```
%Cpu1 : 2.7 us, 0.0 sy, 0.0 ni, 97.3 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 s
```

```
%Cpu2 : 0.0 us, 0.0 sy, 0.0 ni, 0.0 id,100.0 wa, 0.0 hi, 0.0 si, 0.0 s
```

```
%Cpu3 : 5.3 us, 7.9 sy, 0.0 ni, 84.2 id, 0.0 wa, 0.0 hi, 2.6 si, 0.0 s
```

```
MiB Mem : 7863.3 total, 230.4 free, 3847.2 used, 3785.8 buff/cache
```

```
MiB Swap: 8192.0 total, 8191.5 free, 0.5 used. 3191.1 avail Mem
```

```
...
```

zombie数那么高是因为这个 docker container 已经运行20多分钟了。

供大家参考:)

2018-12-09



白华

👍 3

今天重新进行了测试，用了你docker hub上的fix2, fix1, new, new2都进行测试了，发现还是不行，iowait没有升高，平均负载没有上升，没有发现D状态。希望你以后使用centos7系统进行操作，性能会好很多

2018-12-06



悟空

👍 3

真实发现一个问题，系统版本影响太大啦，从第一节平均负载开始，centos6,centos7, 同样的案例，显示的结果不一样，而且，更别说ubuntu了，学起来，比较费劲，因为自己的操作，永远和老师的不一致

2018-12-06





白华

👍 3

老师以后的案例能不能使用centos7系统进行操作？做的很多实验和你的都会有部分偏差，这次偏差更大，相信学习你课程的大部分都是用虚拟机跑的项目，用centos系统使用率会很高，而且实际生产中用centos系统肯定大于Ubuntu，造成的实验偏差会不会也是系统的原因。我也遇到了没有出现D状态的进程，出现了大量Z进程。平均负载并没有提升，反而是下降了。iowait并没有变化。所以恳请您使用centos系统来教学吧

2018-12-05

作者回复

iowait不高是因为案例IO操作不够大导致的。我重新推了一个docker镜像，麻烦再试下看看

2018-12-05



每天晒白牙

👍 2

【D7补卡】

在和老师多次交流下，终于逼得老师发布一个把自己机器跑死的镜像，就可以了，结果和老师的温和了。看到老师之前的例子io压力还是不够啊

```
docker run --privileged --name=app -itd feisky/app:iowait-new2
```

执行这个镜像，iowait打满，直接把我的微信给挤掉了，浏览器都打不开了，不过结果是好的。

继续坚持下去！

2018-12-05



Brown羊羊

👍 2

没有模拟出来系统I/O瓶颈，可以帮忙看下吗：

容器运行起来后只发现一个app进程

```
[root@liyang2 ~]# ps aux|grep /app
```

```
root 23619 0.0 0.0 4368 380 pts/0 Ss+ 17:12 0:00 /app
```

```
root 23777 0.0 0.0 112648 952 pts/0 S+ 17:12 0:00 grep --color=auto /app
```

CPU情况 wa也没有很高

```
%Cpu(s): 1.0 us, 1.5 sy, 0.0 ni, 94.0 id, 3.3 wa, 0.0 hi, 0.2 si, 0.0 st
```

系统：redhat7 3.10.0-327.el7.x86_64 x86_64 GNU/Linux

2018-12-05

作者回复

我的机器配置太弱了，IO已经跑满还是好多人都没有观察到iowait的现象。重新推了一个镜像，加大了IO操作，再试试看现在怎么样

2018-12-05



姜小鱼

👍 2

这个案例的iowait比较高，但是并不影响cpu使用率。因为准确来说，iowait也是属于cpu idle状态的一部分，他和僵尸进程影响的只是平均负载和系统资源

2018-12-05

作者回复

确切的说是CPU繁忙程度，因为iowait也是CPU使用率的一种类型

2018-12-05



耿长学

👍 2

学习了，每天上班路上听听音频看一看，晚上回家整理学习

2018-12-05



songgoogle

👍 1

麻烦换centos吧，更接近实际工作要求

2018-12-07



深蓝

👍 1

同问，关于Uninterruptible sleep(D)状态

的进程如何有效的处理，以前运维的时候遇到过，貌似只能重启机器，不知道还有什么更好的办法

2018-12-07

作者回复

一个基本的思路是要找出进程处于 D 状态的原因，是在等待什么样的I/O资源。比如分析系统调用、进程堆栈等等。

找出根源之后，再去分析这些根源里面到底发生了什么，才导致的没有响应。

当然，也有其他比较hack的方法，但生产环境中不推荐，以免给系统带来未知的损坏。

2018-12-08



mj4ever

👍 1

通过实际操作和资料查阅，本次课程学到了以下知识：

- 1、进程的多种状态，D (Disk Sleep) 状态的进程，会导致平均负载升高
- 2、僵尸进程：
 - (1) 父子进程的运行是异步的过程，父进程需要知道子进程是何时关闭的
 - (2) 子进程需要父进程来收尸，但父进程没有安装SIGCHLD信号处理函数调用wait或waitpid
 - (3) 等待子进程结束，或是子进程执行太快，父进程还没来得及处理子进程状态
 - (4) 父进程退出后，init进程会自动接手子进程，进行清理
 - (5) 僵尸进程已经放弃了几乎所有内存空间，没有任何可执行代码，也不能被调度，仅仅在进程列表中保留一个位置，记载该进程的退出状态等信息供其他进程收集
 - (6) 大量的僵尸进程会用尽 PID 进程号，导致新进程不能创建
- 3、会话和进程组
- 4、实操
 - (1) 查看了僵尸进程 查看僵尸进程的命令 `ps -e -o stat,ppid,pid,cmd | egrep '^[Zz]'` 或 `ps -ef | grep "defunct"`，停止父进程后，僵尸进程会被回收
 - (2) 没有观察到io的问题，可能是和机器配置有关系，不知道是否可以暴露个参数让我们启动docker的时候传参来施加压力。

2018-12-06



ninuxer

👍 1

打卡day8

根据上几天的内容，出现iowait，能想到的分析过程：先用pidstat -u查看进程级别cpu的信息，pidstat -w查看进程级别的自愿中断信息，如果因为io问题，自愿中断应该会飙升，再就是用perf top查看出问题的进程的信息了

2018-12-05



一生一世

👍 1

我的思路是用1、pidstat看看上下文交换情况；2、vmstat看看wa;把僵尸进程的父进程停掉

2018-12-05



Orcsir

👍 0

Flag

2018-12-13



#

👍 0

#man top

20. S -- Process Status

The status of the task which can be one of:

D = uninterruptible sleep

R = running

S = sleeping

T = stopped by job control signal

t = stopped by debugger during trace

Z = zombie

#man ps

PROCESS STATE CODES

Here are the different values that the s, stat and state output specifiers (header "STAT" or "S") will display to describe the state of a process:

D uninterruptible sleep (usually IO)

R running or runnable (on run queue)

S interruptible sleep (waiting for an event to complete)

T stopped by job control signal

t stopped by debugger during the tracing

W paging (not valid since the 2.6.xx kernel)

X dead (should never be seen)

Z defunct ("zombie") process, terminated but not reaped by its parent

For BSD formats and when the stat keyword is used, additional characters may be displayed:

< high-priority (not nice to other users)
 N low-priority (nice to other users)
 L has pages locked into memory (for real-time and custom IO)
 s is a session leader
 l is multi-threaded (using CLONE_THREAD, like NPTL pthreads do)
 + is in the foreground process group
 在ubuntu和centos都没看到idle状态，是内核还是版本问题呢？

2018-12-12

作者回复

应该是内核版本，你的内核是哪个版本的？

2018-12-13



Vicky 🐼🐼🐼

👍 0

老师，我按步骤执行后，ps里面显示exited了，也没有app进程，麻烦老师看一下
 [root@VM_0_5_centos ~]# docker run --privileged --name=app -itd feisky/app:iowait
 t
 c6eaa823f3c7a8c4e29a8925fe424726840b1e2f2a568d22ba89966d8bdfb8fb
 [root@VM_0_5_centos ~]# docker ps -a
 CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
 c6eaa823f3c7 feisky/app:iowait "/app" 3 seconds ago Exited (1) 1 second ago app

2018-12-11

作者回复

执行 docker logs app 看看日志里面报了什么错误？

2018-12-11



Vicky 🐼🐼🐼

👍 0

老师，我执行了docker run --privileged --name=app -itd feisky/app:iowait
 但是，没有app的进程
 [root@VM_0_5_centos ~]# docker run --privileged --name=app -itd feisky/app:iowait
 t
 0cdbd971d7903bb0c289b44a4f7f158f3befa9ec0e9ffec7794a0cacfcad2348
 [root@VM_0_5_centos ~]# ps aux | grep '/app'
 root 7542 0.0 0.0 112648 960 pts/13 R+ 15:25 0:00 grep --color=auto /app

2018-12-11

作者回复

执行 docker logs app 看看报了什么错误？

2018-12-11



Geek_a9ec17

👍 0

[root@instance ~]# docker run --privileged --name=app -itd feisky/app:iowait
 7d1cd41a79cdc71d6c7e8d2882507e2d6eb959d728aa5a7e3c9774c996320a
 [root@instance ~]# docker ps -a

```
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
7d1cd41a79cd feisky/app:iowait "/app" 3 seconds ago Exited (1) 3 seconds ago app
倪老师， docker 镜像run不起来
```

2018-12-11

作者回复

试试源码里的方法 <https://github.com/feiskyer/linux-perf-examples/blob/master/high-iowait-process/README.md>

2018-12-13